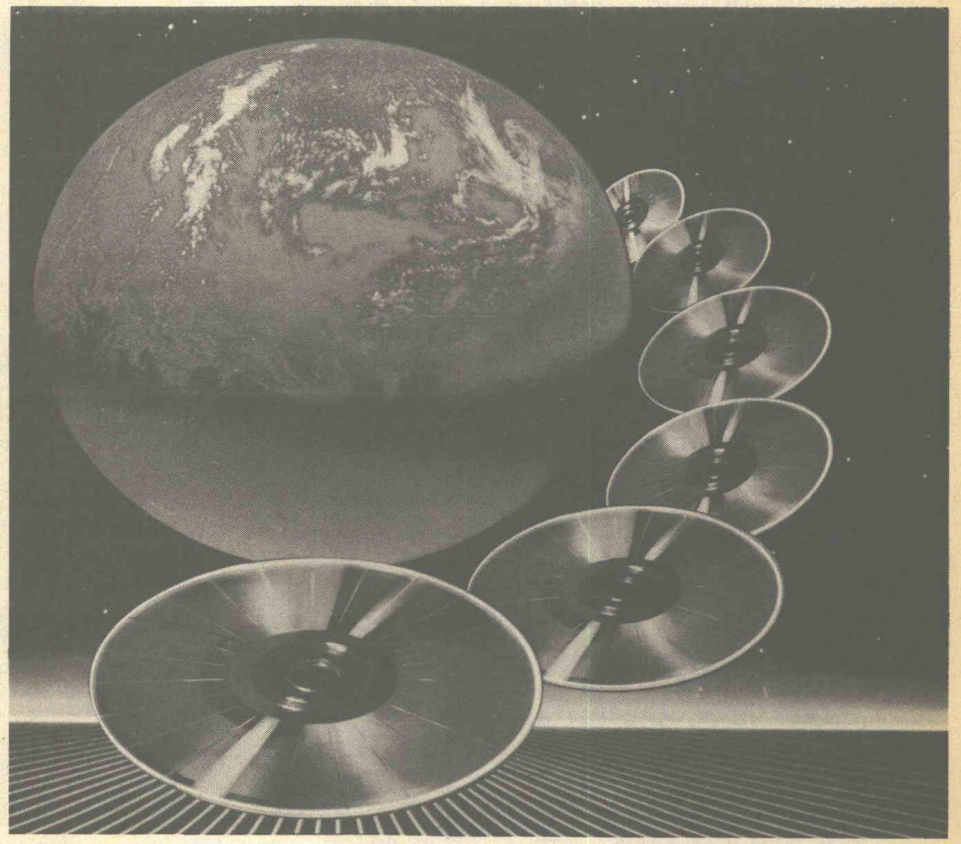
บุญเลิศ เอี่ยมทัศนา

TPW
: ภาพเคลื่อนไหว

กลวิธีในการสร้างภาพกราฟิกที่เคลื่อนไหวได้บนวินโดวส์น่าสนใจและน่าฝึกฝนเพื่อนำไปใช้



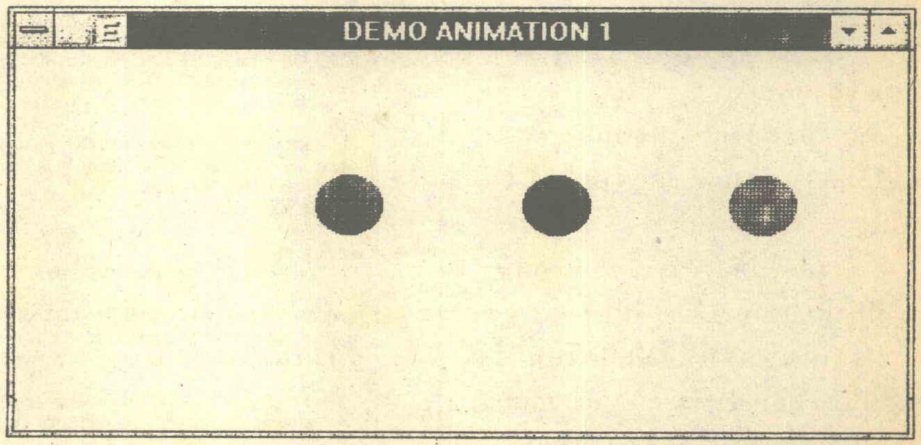
ฉากหนึ่งของ AfterDark ที่ผมชอบมากคือ Aquarium ที่ดูได้ไม่เบื่อพร้อมกับสงสัยไปด้วยว่าเขาทำได้อย่างไร เพราะวินโดวส์ไม่สนับสนุนการเขียนโปรแกรมแสดงภาพเคลื่อนไหว (Animation) และไม่มีฟังก์ชันสำหรับเรื่องนี้ แต่เราก็พอจะเขียนโปรแกรมแสดงภาพเคลื่อนไหวต่าง ๆ ได้บ้าง เช่น การแสดงภาพเลื่อน

โปรแกรมแรก

เพราะความที่ติดใจว่าจะแสดงภาพเคลื่อนไหวบนวินโดวส์ได้อย่างไร พอผู้เขียนเห็นสารบัญของหนังสือเล่มหนึ่งกล่าวถึงเรื่องนี้ก็ดีใจ เป็นโปรแกรมแสดงภาพลูกบอลลอยไปลอยมาในหน้าต่างแต่เมื่อนำมาเขียนโปรแกรมก็ต้องผิดหวัง ซึ่งผิดหวังอย่างไรขอพักไว้ก่อนครับ

โอโอพี

แต่จะนำมาแสดงเพราะผมได้เพิ่มลูกบอลเป็น 3 ลูก ดังในรูปที่ 1 เพื่อแสดงการเขียนโปรแกรมแบบโอโอพี



รูปที่ 1 แสดงภาพลูกบอล 3 ลูก

ตามโปรแกรมในลิสติงที่ 1 ทั้งนี้เพราะผมเห็นว่าการเขียนโปรแกรมบนวินโดวส์ดังที่ได้นำเสนอมานั้น ไม่เป็นการเขียนโปรแกรมแบบโอโอพีที่แท้จริง เป็นเพียงการเขียนโปรแกรมด้วยภาษาที่เป็นโอโอแอล (Object Oriented Language) เมื่อมีโอกาสก็นำมาแสดง และเป็นพื้นฐานของการแสดงภาพเคลื่อนไหวบนวินโดวส์ด้วยโดยที่เมื่อมีลูกบอล 3 ลูก หากเขียนโปรแกรมตามแบบ (ดั้งเดิมที่เคยเขียนกันมาและยังคงใช้ได้ต่อไป) ก็จะต้องกำหนดอะเรย์สำหรับเก็บข้อมูล เช่น ตำแหน่งของลูกบอลแต่ละลูก และจะต้องผ่านค่าไปให้แก่โปรแกรมย่อยที่จะดำเนินการในเรื่องต่าง ๆ เช่น ย้ายตำแหน่งของลูกบอล

แต่เมื่อเขียนโปรแกรมแบบโอโอพี ก็จะมองลูกบอลแต่ละลูกเป็นออปเจ็กต์ (เป็นตัวละคร) แต่ละตัวของสกุลเดียวกันที่สามารถจะส่งสื่อความไปให้ได้ (บอกบทได้) ดังที่ได้สร้างเป็นสกุล BallC ไว้ กล่าวคือ

1. เมื่อต้องการให้กำเนิดออปเจ็กต์ของสกุลนี้คือลูกบอลแต่ละลูก จะต้องผ่านค่าเหล่านี้ไปให้แก่ Init คือ
   - ค่าแฮนเดิลของหน้าต่าง
   - ชื่อไฟล์ที่เก็บภาพบิตแม็พของลูกบอล
   - ตำแหน่งเริ่มต้น (จุดมุมบนซ้าย) ของลูกบอล
2. เมื่อต้องการให้ลูกบอลเลื่อนให้ใช้สื่อความ Move โดยผ่านค่า DC ไปให้
3. เมื่อไม่ต้องการใช้ออปเจ็กต์นี้อีกให้ส่งสื่อความ Done ไปให้

จะเห็นว่าเราเพียงแค่เตรียมค่าเริ่มต้นให้แก่ลูกบอลแต่ละลูก หลังจากนั้นก็เพียงแค่ใช้สื่อความ Move โดยไม่ต้องจดจำรายละเอียดอื่นใดอีก

ลิสติงที่ 1

```
PROGRAM  Animation1;   { animatn1.pas }

{ USES   WinTypes,WinProcs,WObjects,ufidlg; }  { TPW }
USES   WinTypes,WinProcs,Objects,OWindows,ufidlg;   { BPW }

CONST  appname  = 'animatn1';
       thetitle = 'DEMO ANIMATION 1';

       timr1 = 1;   intv = 50;
       rops  = {SrcInvert;}  {SrcCopy;} SrcAnd; { SrcPaint; }

TYPE
  BallCP = ^BallC;
  BallC  = OBJECT(TObject)
    hwndo  : HWnd;
    mveBMP : HBitmap;
    mx,my,dx,dy : Integer;
    mw,mh : LongInt;
 constructor  Init(wnd : HWnd; finame : PChar; xx,yy : Integer);
 destructor   Done; virtual;
 procedure    Move(adc : HDC);
           END;


  WndProCP = ^WndProC;
  WndProC = OBJECT(TWindow)
    ball1,ball2,ball3   : BallC;
    runtmr : Boolean;
 constructor  Init(apar : PWindowsObject; atitle : PChar);
 procedure  SetupWindow;  virtual;
 function   CanClose : Boolean; virtual;
 procedure  Paint(paintdc : HDC;
              var paintInfo : TPaintStruct); virtual;
 procedure  wmTimer(var msg : TMessage);
              virtual wm_First + wm_Timer;
           END;


(* ---------------- BallC Method ---------------------- *)

CONSTRUCTOR  BallC.Init(wnd : HWnd; finame : PChar; xx,yy : Integer);
BEGIN
  TObject.Init;   hwndo := wnd;
  mx := xx;  my := yy;  dx := 1;  dy := 1;
  mveBMP := LoadBitMapFile(finame, mw, mh),
END;


DESTRUCTOR  BallC.Done;
BEGIN
  if  mveBMP <> 0  then  DeleteObject(mveBMP);
  TObject.Done;
END;


PROCEDURE  BallC.Move(adc : HDC);
VAR   wndDC,mveDC : HDC;   res : Bool;  R : TRect;
      omve : THandle;
BEGIN
  if  adc <> 0  then  wndDC := adc
  else  wndDC := GetDC(hwndo);
  GetClientRect(hwndo, R);
  mx := mx + dx;
  if  (mx <= 0) or (mx+mw >= R.right)  then  dx := -dx;
  my := my + dy;
  if  (my <= 0) or (my+mh >= R.Bottom)  then  dy := -dy;

  mveDC := CreateCompatibleDC(wndDC);
  omve  := SelectObject(mveDC, mveBMP);
  BitBlt(wndDC, mx,my,mw,mh, mveDC,0,0, SrcCopy);

  SelectObject(mveDC, omve);  DeleteDC(mveDC);
  if  adc = 0  then  ReleaseDc(hwndo, wndDC);
END;
```

บิตแม็พ

การแสดงภาพเคลื่อนไหว (ซึ่งขอหมายถึงบนวินโดวส์เท่านั้น) ใช้วิธีลอกภาพบิตแม็พดังที่เห็นใน BallC Move จึงขอให้สร้างภาพบิตแม็พเป็นรูปลูกบอลสีแดง, น้ำเงิน และเขียว ขนาดตามต้องการเก็บไว้ในไฟล์ redball.bmp, blueball.bmp และ grnball.bmp ตามลำดับให้อยู่ในไดเรคทอรีเดียวกันกับโปรแกรมนี้

ซึ่งใน WndProC.SetupWindow จะให้ชื่อไฟล์นี้แก่ BallC.Init โหลดให้แก่ mvePMP ของตนและนำออกแสดง ณ ตำแหน่งใหม่ใน BallC.Move

ลบภาพเดิม

สิ่งสำคัญอย่างหนึ่งของการแสดงภาพเคลื่อนไหว คือการลบภาพเดิม ซึ่งในโปรแกรมนี้กระทำโดยวาดภาพใหม่ทับแต่จะทับได้ไม่ทั้งหมด ดังนั้นในการสร้างบิตแม็พซึ่งเป็นการวาดรูปวงกลมตันสีต่าง ๆ ในกรอบสี่เหลี่ยม ซึ่งจะขอเรียกว่ากรอบภาพและในกรณีนี้จะเป็นสี่เหลี่ยมด้านเท่าจะเป็นการวาดภาพวงกลมตันบนพื้นขาวคือ สีเดียวกับพื้นของหน้าต่าง และจะต้องไม่ให้วงกลมสัมผัสกับกรอบภาพ จะต้องเว้นไว้อย่างน้อยด้านละ 1 จุดภาพเมื่อต้องการเลื่อนภาพครั้งละ 1 จุดภาพ ตามค่าที่ให้แก่ dx และ dy ใน BallC.Init

นั่นคือเมื่อให้แสดงภาพบิตแม็พในตำแหน่งใหม่จะลบภาพวงกลมเดิมได้ทั้งหมด ส่วนพื้นนอกวงกลมซึ่งความจริงจะเป็นสี่เหลี่ยมเสี้ยวเล็ก ๆ จะไม่ถูกลบและไม่ต้องลบด้วย เพราะสีเดียวกับพื้นหน้าต่าง

และนี่คือเหตุที่ทำให้ผมผิดหวัง เพราะในการสร้างภาพเคลื่อนไหวนั้นหากเราวาดรูปลูกบอลบนพื้นสีอื่น เช่น สี

ลิสติงที่ 1 (ต่อ)

```
(* ------------------ WndProC Method ------------------ *)

CONSTRUCTOR WndProC.Init(apar : PWindowsObject; atitle : PChar);
BEGIN
  TWindow.Init(apar,atitle);  runtmr := false;
  attr.x := 10;  attr.y := 10;  attr.w := 600;  attr.h := 400;
END;


PROCEDURE WndProC.SetupWindow;
VAR  res : Word;  patres : Bool;  adc,sveDC : HDC;
     hold : THandle;  wndDC : HDC;  mainfi : PChar;
BEGIN
  TWindow.SetupWindow;
  ball1.Init(HWindow, 'redball.bmp', 100,50);
  ball2.Init(HWindow, 'blueball.bmp', 200,50);
  ball3.Init(HWindow, 'grnball.bmp', 300,50);
END;


PROCEDURE WndProC.Paint(paintdc : HDC; var paintInfo : TPaintStruct);
VAR
      srcDC : HDC;  hold1,hold2 : THandle;  patres : Bool;
BEGIN
  if runtmr then KillTimer(HWindow, timr1);
  ball1.Move(paintdc);  ball2.Move(paintdc);  ball3.Move(paintdc);
  SetTimer(HWindow, timr1, intv, nil);  runtmr := true;
END;


FUNCTION WndProC.CanClose : Boolean;
BEGIN
  if runtmr then KillTimer(HWindow, timr1);
  ball1.Done;  ball2.Done;  ball3.Done;
  CanClose := TWindow.CanClose;
END;


PROCEDURE WndProC.wmTimer(var msg : TMessage);
VAR  R : TRect;
BEGIN
  ball1.Move(0);  ball2.Move(0);  ball3.Move(0);
END;


(* ------------------ TheAppC Class ------------------ *)

TYPE
  TheAppC = OBJECT(TApplication)
    procedure InitMainWindow; virtual;
            END;


PROCEDURE TheAppC.InitMainWindow;
BEGIN
  MainWindow := New(WndProCP, Init(nil,thetitle));
END;

(* ------------------ MAIN PROGRAM ------------------ *)

VAR  app : TheAppC;

BEGIN
  app.Init(appname);
  app.Run;
  app.Done;
END.
```

■

เหลือง ก็จะทิ้งรอยสีเหลืองไว้เป็นทาง ทำให้แสดงภาพเคลื่อนไหวบนพื้นสีต่าง ๆ ไม่ได้ เช่น ให้ฉากหลังเป็นรูปวิว จึงไม่ใช่วิธีที่ใช้งานได้ทั่วไป

ไทม์เมอร์

การนำงานของโปรแกรมนี้ใช้ไทม์-เมอร์ (timer) 1 ตัว มีกิจกรรมเกี่ยวข้องกับเรื่องนี้ 2 กิจกรรมคือ WndProc. Paint กับ WndProC.wmTimer โดยที่ Paint เป็นผู้ตั้งเวลาด้วย SetTime ตามค่าที่ผ่านไปให้ดังนี้

– HWindow คือค่าแฮนเดิลของหน้าต่าง

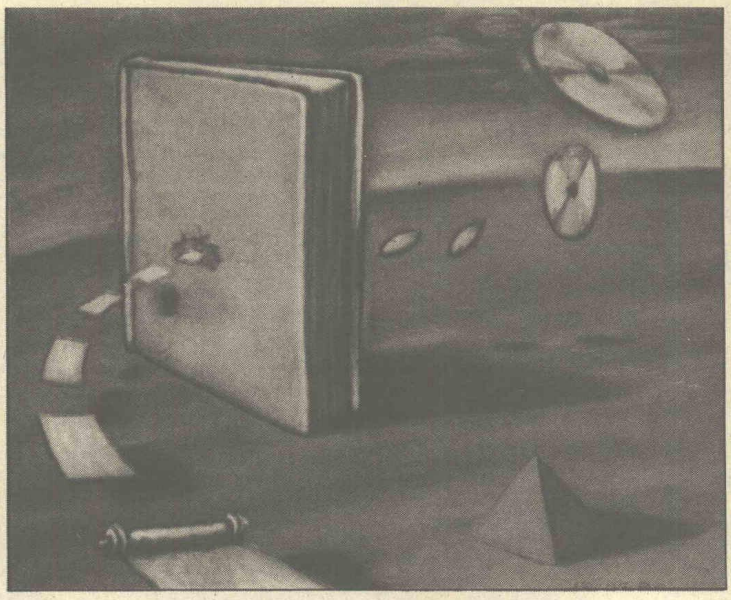
– timr1 เป็น ID ที่เราตั้งขึ้น จุดประสงค์เพื่อตรวจสอบเมื่อใช้ไทม์เมอร์หลายตัว

– intv คือค่าช่วงเวลาที่ต้องการให้ค่าเป็นมิลลิวินาที ค่าที่ให้คือ 50 เป็นค่าที่เล็กสุดตามที่จะกล่าวต่อไป

– nil อาจให้เป็นแอดเดรสของโปรแกรมย่อยที่จะรับทราบการอินเตอร์รัพต์ของไทม์เมอร์ ถ้าให้เป็น nil หมาย-ถึง ให้วินโดวส์แจ้งให้ทราบทางเมสเสจ wm-Timer

ตามที่กล่าวว่า เพื่อรับทราบการอินเตอร์รัพต์ของไทม์เมอร์ ขอทำความเข้าใจให้ชัดเจนว่าจะไม่มีการอินเตอร์รัพต์ใด ๆ เข้ามาในโปรแกรมที่เขียนบนวินโดวส์ (และเราจะต้องไม่ให้มีด้วย) ในเรื่องของไทม์เมอร์วินโดวส์จะใช้ไทม์เมอร์อินเตอร์รัพต์ของไบออส ซึ่งจะอินเตอร์รัพต์เข้ามาทุก 51.925 มิลลิ-วินาทีเป็นฮาร์ดแวร์อินเตอร์รัพต์

เมื่อเราตั้งค่าเวลาด้วย SetTimer จะเป็นการแจ้งให้วินโดวส์ทราบไว้เท่านั้น โดยวินโดวส์จะนำค่าเวลาที่เราให้ไปมาหารด้วย 54.925 ในแบบปัดเศษทิ้ง แต่ถ้าได้ค่าเป็น 0 จะให้เป็น 1 เป็นจำนวน



ครั้งที่จะนับครั้งของการอินเตอร์รัพต์ เมื่อครบกำหนดจะแจ้งให้เราทราบทางโปรแกรมย่อยหรือ wm_Timer ดังกล่าวมา

ดังนั้นเมื่อเราให้ค่าเป็น 0 หรือเป็น 100 จะได้ค่าเวลาเท่ากันคือ 1 ครั้ง แต่การแจ้งช่วงเวลานี้วินโดวส์อาจไม่แจ้งทันทีหากคิดงานอื่นอยู่ก็จะระงับไว้ก่อน เราจึงอาจไม่ได้ช่วงเวลาตามที่กำหนดเสมอไป แต่ก็เป็นวิธีเดียวที่ใช้ในการเขียนโปรแกรมบนวินโดวส์เมื่อต้องการทำงานเป็นวงรอบรอเวลา

การใช้งานไทม์เมอร์เมื่อไม่ต้องการใช้แล้วจะต้องส่งคืน เพราะวินโดวส์มีไทม์เมอร์อยู่ 32 ตัว (มีเรคอร์ดที่จะเก็บค่าเวลาตามที่แอพพลิเคชันต่าง ๆ จะขอตั้งอยู่ 32 เรคอร์ด) เท่านั้น ในโปรแกรมนี้เราใช้ตลอดเวลาจึงส่งคืนใน WndProC.Canelose ด้วย KillTimer

เช่นเดียวกันเมื่อเราต้องการหยุดเวลาคือใน WndProC.Paint ก็จะต้องหยุดด้วย KillTimer เป็นการส่งคืนและขอมาตั้งใหม่ด้วย SetTime

wm_Timer

เมื่อเราตั้งเวลาโดยขอให้วินโดวส์แจ้งทางเมสเสจ เมื่อถึงช่วงเวลาที่กำหนดวินโดวส์จะส่งเมสเสจ wm_Timer แจ้งให้ทราบ มีค่า ID ตามที่เรากำหนดอยู่ใน msq.wParam แต่เนื่องจากในโปรแกรมนี้ใช้ไทม์เมอร์เพียง 1 ตัว ในโปรแกรมของกิจกรรม WndProC.-wmTimer จึงไม่ได้นำมาแยก คงเพียงแต่ส่งสื่อความให้ลูกบอลทั้งสามเคลื่อนที่

ทดลองรัน

เมื่อรันโปรแกรมนี้จะเห็นภาพลูกบอลทั้งสามวิ่งอยู่ในหน้าต่างในลักษณะลอยตัว เมื่อกระทบกับกรอบหน้าต่างจะกระดอนในลักษณะมุมตกเท่ากับสะท้อน ก็พอดูได้ครับและผมหมายถึงภาพเคลื่อนไหวทั้งหมดที่วาดด้วยวิธีนี้ไม่ว่าจะให้มีการกระดอนหรือไม่

หากต้องการให้ลูกบอลวิ่งเร็วขึ้นต้องเพิ่มระยะทางที่จะให้เลื่อนในแต่ละครั้ง คือเพิ่มค่าให้แก่ dx และ dy (ต้องเพิ่มช่องว่างในบิตแม็พด้วย) หากต้องการให้วิ่งช้าลงต้องเพิ่มช่วงเวลาเช่นให้ค่า intv เป็น 110 แต่การเคลื่อนตัวจะเป็นในลักษณะกระตุกอย่างเห็นเด่นชัด

ภาพเลื่อนบนฉากหลัง

หากต้องการแสดงภาพเลื่อนบนพื้นที่เป็นฉากหลัง จะต้องใช้โปรแกรม Animation 2 ตามลิสติงที่ 2 จะได้ผลการรันตามรูปที่ 2 กล่าวคือ

ภาพฉากหลัง

หากคุณมีภาพฉากหลังสวย ๆ เก็บไว้ในไฟล์ .BMP ให้นำมาใช้ได้ครับ โดยให้ชื่อไฟล์แก่ mainfi ตามที่เห็นอยู่ตอนต้นของโปรแกรม แต่ตามรูปที่ 2 ผมใช้ภาพในหน้าต่างของเทอร์โบปาสคาลเป็นฉากหลัง โดยดำเนินการดังนี้

– จัดหน้าต่างเอกสาร คือ หน้าต่างของไฟล์ animatn 1.pas กับ animatn2.-pas ให้เป็นดังในรูป คือ เหลือพื้นที่ทางด้านซ้ายไว้

– ใช้โปรแกรม ScreenCaptor (ตามที่กล่าวในฉบับก่อน) เก็บภาพตรงส่วนที่ต้องการ แล้วเก็บภาพลงคลิป-บอร์ดด้วย File/Put BMP ซึ่งคุณอาจจะให้คีย์ Alt+Print Screen ได้โดยตรงแต่จะได้ภาพทั้งหน้าต่างเทอร์โบปาสคาล

– เปิดใช้ Paintbrush แล้วปะด้วยรายการ Edit/Paste ระบายสีเขียวในกรอบทางด้านซ้าย (ตามที่ในรูปเห็นเป็นพื้นสีเทา) เก็บลงไฟล์ backgrnd.-bmp

ตามรูปในกรอบสีเขียว (คือสีเทา) จะมีกรอบสี่เหลี่ยมเล็ก ๆ อีก 2 กรอบ เป็นผลของการรันซึ่งเป็นเรื่องที่เพิ่มขึ้นมา ขอให้ถือว่าไม่มีไว้ก่อนครับ

ตัวภาพเลื่อน

สำหรับภาพที่จะให้เป็นภาพเลื่อน ผมทำเป็นรูปห่วงสีแดง หรือจะเรียกว่าเป็นวงแหวนได้เช่นกัน ตามรูปจะเห็นเป็นวงสีดำอยู่ทางด้านขวาตรงบรรทัด USES ทับคำว่า WinTypes อยู่ บนจอ

ลิสติงที่ 2

```
PROGRAM  Animation2;   { animat2.pas }

{ USES   WinTypes,WinProcs,WObjects,ufidlg; }  { TPW }
USES   WinTypes,WinProcs,OWindows,ufidlg;      { BPW }

CONST  appname  = 'animatn2';
       thetitle = 'DEMO ANIMATION 2';

       mainfi = 'backgrnd.bmp';
       timr1  = 1;    intv = 50;

TYPE
  WndProCP = ^WndProC;
  WndProC = OBJECT(TWindow)
    mainBMP,mveBMP,negatBMP : HBitmap;
    mx,my,dx,dy,delay : Integer;
    mainw,mainh,mw,mh : LongInt;
    timeon : Word;
 constructor  Init(apar : PWindowsObject; atitle : PChar);
 function   CanClose : Boolean; virtual;
 procedure  Paint(paintdc : HDC;
              var paintInfo : TPaintStruct); virtual;
 procedure  wmRButtonDown(var msg : TMessage);
             virtual wm_First + wm_RButtonDown;
 procedure  wmTimer(var msg : TMessage);
              virtual wm_First + wm_Timer;
 procedure  Move(adc : HDC);
            END;


(* ------------------ WndProC  Method ----------------- *)

CONSTRUCTOR  WndProC.Init(apar : PWindowsObject; atitle : PChar);
BEGIN
  TWindow.Init(apar,atitle);
  attr.x := 10;  attr.y := 10;  attr.w := 600;  attr.h := 400;
  mx := 500;  my := 150;  timeon := 0;
  dx := 1;  dy := 1;  delay := -1;
  mainBMP := LoadBitmapFile(mainfi, mainw, mainh);
  negatBMP := LoadBitMapFile('redring0.bmp', mw, mh);
  mveBMP := LoadBitMapFile('redring.bmp', mw, mh);
END;


FUNCTION  WndProC.CanClose : Boolean;
BEGIN
  if  timeon <> 0  then  KillTimer(HWindow, timr1);
  if  mainBMP <> 0  then  DeleteObject(mainBMP);
  if  mveBMP <> 0  then  DeleteObject(mveBMP);
  if  negatBMP <> 0  then  DeleteObject(negatBMP);
  CanClose := TWindow.CanClose;
END;


PROCEDURE  WndProC.Paint(paintdc : HDC; var paintInfo : TPaintStruct);
VAR
      srcDC : HDC;  hold1,hold2 : THandle;  patres : Bool;
BEGIN
  if timeon <> 0  then  KillTimer(HWindow, timr1);
  DispBMPs(HWindow, mainBMP, 0,0,mainw,mainh, false,
                             0,0,mainw,mainh);
  Move(paintdc);
  timeon := SetTimer(HWindow, timr1, intv, nil);
 { timeon := 0; }
END;


PROCEDURE  WndProC.wmTimer(var msg : TMessage);
VAR   R : TRect;
BEGIN
  GetClientRect(HWindow, R);
  KillTimer(HWindow, timr1);
```

ลิสติงที่ 2 (ต่อ)

```
  Move(0);
  timeon := SetTimer(HWindow, timr1, intv, nil);
END;


PROCEDURE  WndProC.wmRButtonDown(var msg : TMessage);
BEGIN
  if  timeon = 0  then
    timeon := SetTimer(HWindow, timr1, intv, nil);
END;


PROCEDURE  WndProC.Move(adc : HDC);
VAR   wndDC,mainDC,mveDC,pokeDC,negatDC : HDC;   res : Bool;
      pokeBMP : HBitmap;  R : TRect;
      omain,omve,opoke,obg,onegat : THandle;
BEGIN
  GetClientRect(HWindow, R);
  mx := mx + dx;
  if  (mx <= 0) or (mx+mw >= R.right)  then  dx := -dx;
  my := my + dy;
  if  (my <= 0) or (my+mh >= R.Bottom)  then  dy := -dy;

  if  adc <> 0  then  wndDC := adc
  else  wndDC := GetDC(HWindow);
  mainDC := CreateCompatibleDC(wndDC);
  omain  := SelectObject(mainDC, mainBMP);

  pokeBMP := CreateCompatibleBitmap(wndDC, mw, mh);
  pokeDC := CreateCompatibleDC(wndDC);
  opoke := SelectObject(pokeDC, pokeBMP);
  mveDC := CreateCompatibleDC(wndDC);
  omve  := SelectObject(mveDC, mveBMP);
  negatDC := CreateCompatibleDC(wndDC);
  onegat  := SelectObject(negatDC, negatBMP);

  BitBlt(pokeDC, 0,0,mw,mh, mainDc,mx,my, SrcCopy);
  BitBlt(pokeDC, 0,0,mw,mh, negatDC,0,0, SrcPaint);
BitBlt(wndDC, 50,60,mw,mh, pokeDC,0,0, SrcCopy);
  BitBlt(pokeDC, 0,0,mw,mh, mveDC,0,0, SrcAnd);
BitBlt(wndDC, 150,60,mw,mh, pokeDC,0,0, SrcCopy);
  BitBlt(wndDC, mx,my,mw,mh, pokeDC,0,0, SrcCopy);


  SelectObject(mainDC, omain);  DeleteDC(mainDC);
  SelectObject(pokeDC, opoke);  DeleteDC(pokeDC);
  SelectObject(mveDC, omve);  DeleteDC(mveDC);
  SelectObject(negatDC, onegat);  DeleteDC(negatDC);
  DeleteObject(pokeBMP);
  if  adc = 0  then  ReleaseDc(HWindow, wndDC);
END;


(* ------------------  TheAppC Class  ---------------- *)

TYPE
  TheAppC = OBJECT(TApplication)
    procedure  InitMainWindow; virtual;
            END;


PROCEDURE  TheAppC.InitMainWindow;
BEGIN
  MainWindow := New(WndProCP, Init(nil,thetitle));
END;

(* --------------------  MAIN PROGRAM  ------------------ *)

VAR   app :  TheAppC;

BEGIN
  app.Init(appname);
  app.Run;
  app.Done;
END.
```

ภาพจะเห็นเป็นภาพห่วงลอยอยู่เหนือฉากหลัง

เพื่อให้ได้ภาพเคลื่อนไหวดังกล่าวจะต้องเตรียมไฟล์บิตแม็พ 2 ไฟล์สำหรับรูปที่จะให้เคลื่อนไหว 1 รูป สำหรับรูปนี้ผมได้ดำเนินการใช้บิตแม็พเอดิเตอร์ เช่น WRT หรือ Resource Workshop เพื่อกำหนดขนาดของแผ่นภาพ (ในตัวอย่างแรกก็เช่นกัน) กล่าวคือ

– กำหนดขนาดของแผ่นภาพเป็น 40×40 จุดภาพ จำนวนสี 16 สี (ตามขีดความสามารถของจอภาพวีจีเอธรรมดา)

– วาดวงกลมแดงโดยเลือกขนาดของปากกาให้ใหญ่พอควรบนพื้นสีขาว คือ ไม่ต้องระบายสีใดแล้วเก็บลงไฟล์ให้ชื่อว่า redring.bmp

– ระบายสีพื้นหลังทั้งนอกและในห่วงด้วยสีดำ แล้วเก็บลงไฟล์ให้ชื่อว่า redrisg0.bmp

การทำแผ่นใส

การแสดงภาพเลื่อนตามตัวอย่างแรก เทียบได้กับการวาดรูปลูกบอลสีต่าง ๆ บนกระดาษขาวแล้วนำไปเลื่อนบนกระดาษขาว แต่ตามโปรแกรมใหม่นี้ต้องการวาดรูปห่วงให้เทียบได้กับการวาดบนแผ่นใส ก็ต้องขอกล่าวไว้ก่อนว่าเราไม่สามารถทำแผ่นใสถาวรได้ จะทำได้เฉพาะเมื่อเราจะนำรูปห่วงไปลงแต่ละครั้ง คือทุกครั้งที่จะเลื่อนห่วงไปต้องทำแผ่นใสใหม่

จะทำได้โดยวิธีที่เรียกว่าการพรุน หรือการเจาะรู (grave) เพื่อจะนำรูปที่ต้องการ คือ รูปห่วงไปฝังคือให้เราสร้างภาพบิตแม็พ 2 ภาพ ภาพแรกเป็นภาพตามปกติ อีกภาพหนึ่งให้ระบายพื้นตรงที่ต้องการให้ใสด้วยสีดำ ดังที่ได้ดำเนินการไว้แล้ว

ก็ต้องขอย้อนกลับไปกล่าวถึงการ

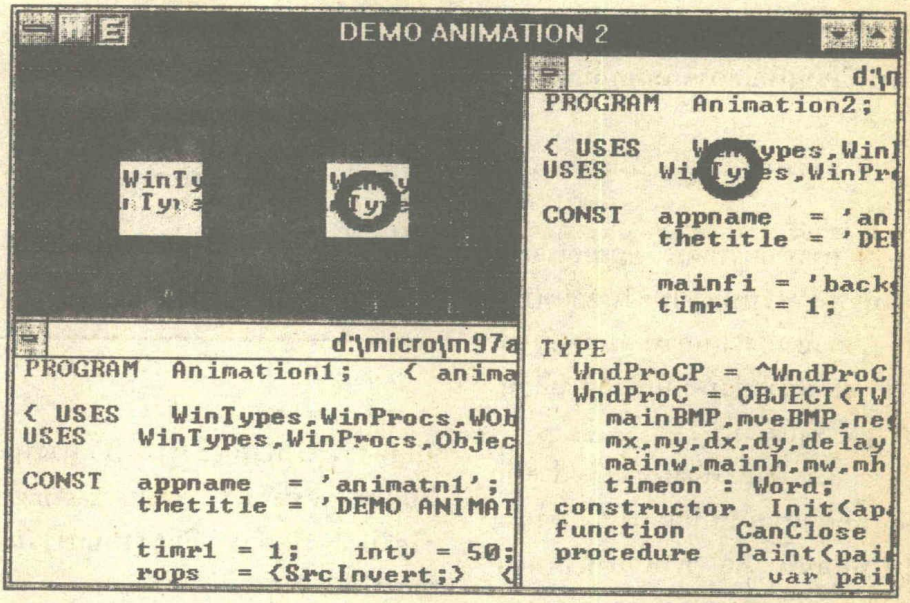
เก็บค่าสีในบิตแม็พซึ่งจะขอเรียกว่าเก็บเป็นหมายเลขสี เมื่อให้แสดงสีได้ 16 สี จุดหนึ่ง ๆ เทียบได้กับจุดภาพหนึ่ง ๆ จุดมีค่าสีได้ตั้งแต่ 0 ถึง 15 โดยที่สีดำจะมีค่าเป็น 0 และสีขาวมีค่าเป็น 15

การพรุน คือการ OR ภาพพื้นหลังด้วยแผ่นภาพที่มีพื้นสีดำ ดังนั้นภาพพื้นหลังส่วนที่ถูก OR ด้วยสีดำจึงเทียบได้กับว่าไม่ได้ถูก OR แต่ส่วนที่ถูก OR ด้วยรูปห่วงสีแดงเทียบได้กับถูกพรุน เช่นสมมติว่าจุดหนึ่งของภาพพื้นหลังมีค่าสีเป็น 2 คือ 0010 b (b หมายถึงเลขฐานสอง) และสมมติว่าสีแดงมีค่าเป็น 5 คือ 0101 b (จะให้เป็นค่าใดก็ได้เพราะเราอาจใช้ห่วงสีอื่นได้) หลังการ OR จะได้ค่าสีของจุดนี้เป็น 0111 b เทียบได้กับถูกเจาะรูให้กว้างออก

การฝัง คือการ AND ภาพพื้นหลังด้วยแผ่นภาพปกติ ดังนั้นภาพพื้นหลังที่ถูก AND ด้วยสีขาวจึงเทียบได้กับว่าไม่ได้ถูก AND ดังนั้นภาพส่วนนี้หลังจาก OR และ AND แล้วจึงคงเดิมเทียบได้กับถูกวาดทับด้วยสีใส แต่ส่วนที่ถูก AND ด้วยสีแดง เช่นตามตัวอย่างที่ยกมาคือ 0111 b AND 0101 b จะได้ค่าเป็น 0101 b คือสีแดง เทียบได้กับการฝังสีแดง จึงได้ภาพโดยส่วนรวมเป็นห่วงสีแดงบนพื้นสีต่าง ๆ

ตัวโปรแกรม

โปรแกรมของโปรแกรม Animation2 อยู่ในลิสติงที่ 2 ซึ่งคล้ายกับโปรแกรม Animation1 มีข้อแตกต่างใน Move ซึ่งนำมาไว้ในโปรแกรมของหน้าต่างหลักเป็น WndProC.Move โดยที่ทั้ง Paint และ wmTimer เรียกใช้โปรแกรมนี้ แตกต่างกันตรง Paint จะแสดงภาพฉากหลังใหม่ก่อน ส่วน wmTiver ใช้ภาพที่แสดงอยู่ในขณะนั้น ใน Move จึงได้ดำเนินการดังนี้ คือ



รูปที่ 2 ผลการรันลิสติงที่ 2

1. คำนวณหาตำแหน่งใหม่ของแผ่นภาพพื้นหน้าให้แก่ mx และ my แผ่นภาพพื้นหน้า คือ แผ่นภาพของภาพบิตแม็พของห่วง (ซึ่งจะมีทั้งพื้นขาวและพื้นดำ)

2. ทำแผ่นภาพบิตแม็พต่าง ๆ ดังนี้

– นำภาพบิตแม็พของฉากหลังที่เก็บไว้ใน mainBMP มาทำเป็นแผ่นภาพฉากหลังใน mainDC

– สร้างแผ่นภาพใหม่ให้มีขนาดเท่ากับแผ่นภาพพื้นหน้าใน pokeDC ซึ่งจะขอเรียกว่าแผ่นภาพสำหรับปะ

– นำภาพบิตแม็พของห่วงพื้นขาวและพื้นดำที่เก็บไว้ใน moveBMP และ negatBMP มาทำเป็นแผ่นภาพพื้นหน้าพื้นขาวและพื้นดำใน moveDC และ negatDC ตามลำดับ

3. ทำแผ่นใสและนำออกแสดง (ในโปรแกรมก่อนนี้ จะมีโปรแกรมที่ใช้ฟังก์ชัน BitBlt อยู่ 2 บรรทัดที่เยื้องออกมาผิดปกติ การบรรยายในข้อนี้จะขอเว้น 2 บรรทัดนี้ไว้ก่อน) กล่าวคือ

– ให้ลอกภาพจากแผ่นภาพฉากหลังตรงที่จะนำภาพพื้นหน้าไปปะให้แก่แผ่นภาพสำหรับปะ

– นำแผ่นภาพสำหรับปะมา OR กับแผ่นภาพพื้นหน้าพื้นดำ (SreInvert ให้ลอกด้วยการ OR)

- นำแผ่นภาพสำหรับปะมา AND กับแผ่นภาพพื้นหน้าพื้นขาว (SreAnd ให้ลอกภาพด้วย AND)

– ลอกภาพในแผ่นภาพสำหรับปะไปออกจอภาพ เป็นการปะภาพ ณ ตำแหน่งใหม่

โปรแกรมตามที่กล่าวมานี้ อาจจะไม่ตรงกับที่กล่าวในหัวข้อการทำแผ่นใสอยู่บ้าง คือ จะกลายเป็นลอกภาพจากต้นฉบับมาปะด้วยแผ่นใส แล้วนำไปปะบนจอภาพอีกทอดหนึ่ง ซึ่งก็เป็นไปตามวิธีการเขียนโปรแกรมครับ

การคืนภาพฉากหลัง

ความจริงแล้วการแสดงภาพเลื่อนจะต้องประกอบด้วย การทำแผ่นใส การลบภาพเดิมและการคืนภาพฉากหลัง

ในโปรแกรมนี้ก็ประกอบด้วยกรรมวิธีทั้งสามนี้เช่นกันแต่อาจกลมกลืนกันอยู่ กล่าวคือ

– การทำแผ่นใส เป็นไปตามที่กล่าวมา

– การลบภาพเดิม คือเมื่อเรานำแผ่นภาพสำหรับปะไปปะในตำแหน่งใหม่ จะเป็นการลบภาพเดิมแต่ลบไม่หมด เหลือบางส่วนเป็นเสี้ยวสีเหลี่ยมเล็ก ๆ ดังที่กล่าวมา

– การคืนภาพฉากหลัง ส่วนที่เหลือนี้ก็คือส่วนที่เป็นแผ่นใส ดังนั้นภาพในส่วนนี้ คือ ภาพของฉากหลังตรงนั้น จึงเป็นการคืนภาพฉากหลังให้โดยอัตโนมัติ

ภาพเสริม

โปรแกรมสำหรับแสดงภาพเลื่อนมีเพียงเท่านี้ แต่มีโปรแกรมอยู่ 2 บรรทัดใน Move ที่ผมยังไม่ได้กล่าวถึง บรรทัดแรกจะนำภาพที่ OR แล้วไปแสดงตรงมุมบนซ้ายของหน้าต่าง (ในกรอบสีเขียว) ส่วนบรรทัดที่สองจะนำภาพที่ AND แล้วไปแสดงทางขวาของภาพแรก ทั้งนี้เพื่อ

- ให้เห็นภาพที่เกิดจากการ OR และ AND
- ให้เห็นการแสดงภาพอีกแนวทางหนึ่ง โดยไม่ต้องคิดถึงคือไม่ต้อง OR หรือ AND เพื่อคุณจะเกิดแนวความคิดในการนำไปใช้

วงรอบการรัน

บางท่านอาจสงสัยว่าจะนำวงรอบเวลามาใช้แทนไทม์เมอร์ได้หรือไม่ คำตอบคือ ไม่ได้ครับหากให้ทำงานอยู่ในวงรอบเวลาตลอดเวลา เพราะจะทำให้ผู้ใช้ปิดหน้าต่างยุติการรันโปรแกรมเราไม่ได้ คำอธิบายในเรื่องนี้จะเป็นคำตอบที่แสดงว่าวินโดวส์ (ถึงรุ่น 3.1) ทำงานในแบบมัลติทาสกิง (multi tasking) หรือไม่เพียงใดด้วย ถือวงรอบการรันตามรูปที่ 3

```
while GetMessage(msg, 0, 0, 0)  do
begin
  TranslateMessage(msg);
  DispatchMessage(msg);
end;
```

รูปที่ 3
วงรอบการรัน

โปรแกรมส่วนนี้ก็คือตรง app.Run หรือถ้าเขียนด้วยภาษา C จะอยู่ใน WinMain คือ เมื่อขึ้นทะเบียนหน้าต่าง ให้กำเนิดหน้าต่าง และให้แสดงหน้าต่างแล้ว จะมาทำงานที่วงรอบนี้จนกว่าผู้ใช้จะยุติการรัน จึงขอเรียกว่าวงรอบการรัน ซึ่งได้มีการดำเนินการดังนี้ คือ

1. ถอนเมสเสจจากคิวของตนด้วย GetMessage
2. ส่งไปให้วินโดวส์ตีความด้วย TranslateMessage
3. ส่งไปให้แก่หน้าต่างที่แอคทีฟอยู่ในขณะนั้นด้วย DispatchMessage (แอพพลิเคชันหนึ่ง ๆ อาจมีหน้าต่างได้หลายบาน)

ฟังก์ชันทั้งสามนี้เป็นของวินโดวส์ และตรงนี้ครับที่วินโดวส์จะสอดแทรกเข้ามาควบคุมการรันของโปรแกรมได้โดยที่

– ถ้าไม่มีเมสเสจในคิว วินโดวส์จะนำเวลาทั้งหมดไปใช้

– ถ้ามีเมสเสจในคิว วินโดวส์จะให้แต่ละแอพพลิเคชันที่รันอยู่ในขณะนั้นทำงานเป็นช่วง ๆ ตามค่า time slice โดยแบ่งเฉลี่ยเวลากัน

สถานะของโปรแกรม

ทดลองดูได้โดยรันโปรแกรมทั้งสองนี้จะเห็นทั้งลูกบอลและห่วงเคลื่อนตัววิ่งไปมา แสดงว่าโปรแกรมทั้งสองนี้รันในแบบมัลติทาสกิง แต่มีเงื่อนไขหลายประเด็นครับ คือ

1. ถ้าผู้ใช้คลิ๊กเม้าส์ที่ไตเติลบาร์เช่นเพื่อจะเลื่อนหน้าต่าง ทั้งลูกบอลและห่วงจะหยุดเคลื่อนตัว แสดงว่าเมื่อเป็นงานของวินโดวส์ วินโดวส์จะนำเวลาไปใช้ทั้งหมด และโปรแกรมทั้งสองนี้คงจะต้องเรียกว่าอยู่ในสถานะที่ไม่รัน เพราะวินโดวส์ไม่ให้ถอนเมสเสจจากคิว

2. แอพพลิเคชันใดที่ไม่แอคทีฟจะไม่ได้รับเมสเสจจากคีย์บอร์ดและเม้าส์ คงได้รับเมสเสจบางเมสเสจจากวินโดวส์เช่น wm__Paint ดังนั้นถึงรันอยู่ก็เทียบกับว่าไม่ได้รัน

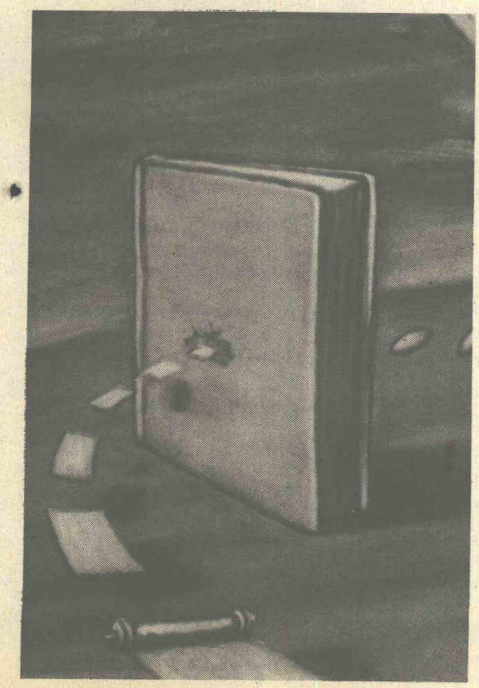
3. ถ้าในโปรแกรมของเราทำงานตามเมสเสจที่ได้รับเนิ่นนานเกินกว่าเวลาที่กำหนดไว้เป็น time slice (เช่น 20 มิลลิวินาที) time slice ก็จะไม่เป็น time slice และที่ร้ายคือถ้าทำงานเป็นวงรอบไม่ยอมกลับมาทำงานที่วงรอบการรัน วินโดวส์จะทำอะไรไม่ได้เลยรวมทั้งยุติการรันของโปรแกรมนี้ก็ไม่ได้ดังจะแสดงในโปรแกรม IamBig

โปรแกรม IamBig

ขอให้ลอกโปรแกรม Animation1 ตามลิสติงที่ 1 ให้ไฟล์ใหม่ชื่อ iambig.pas เปลี่ยนชื่อโปรแกรมเป็น IamBig พร้อมด้วยชื่อต่าง ๆ ที่เกี่ยวข้อง แล้วเพิ่มกิจกรรม wmRbultonDown ตามลิสติงที่ 3

เมื่อรันโปรแกรมนี้ผลการรันคงเป็นเช่นเดียวกับโปรแกรม Animation1 และขอให้รันโปรแกรม Antimation2 มาคู่กันด้วย จะคงรันตามปกติเช่นกัน แต่ถ้าคุณคลิ๊กเม้าส์ขวาใน IamBig เมื่อหน้าต่างนี้แอคทีฟ ทั้งลูกบอลและห่วงจะหยุดเคลื่อนไหวทันทีมีแต่เสียงปิ๊บ ๆ และคุณจะดำเนินการใดไม่ได้เลยไม่ว่าจะใช้คีย์บอร์ดหรือเม้าส์ และไม่ว่ากับหน้าต่างหรือไอคอนใด ประมาณ 30 วินาที

ทั้งนี้เพราะว่าใน wmRBulton-Down ได้ทำวงรอบเวลารอบในคือ FOR ไว้รอบละประมาณ 1 วินาทีแล้วให้ REPEAT พับ 30 ซึ่งระหว่างนี้จะ



ลิสติงที่ 3

```
procedure  wmRButtonDown(var msg : TMessage);
           virtual wm_First + wm_RButtonDown;


PROCEDURE  WndProC.wmRButtonDown(var msg : TMessage);
VAR   a,b : Real;  cnt,num : LongInt;  eternity : Boolean;
BEGIN
  a := 0;  b := 0; num := 0;  eternity := false;
  repeat
    for  cnt := 1 to 60000  do
      begin  if a <> 1  then  a := 0.123456789;
             if b <> 1  then  b := 0.987654321;
             a := a * b  end;
    MessageBeep(0);  inc(num);
  until  num > 30;
END;
```

ไม่ไปทำงานที่วงรอบการรันขาดการติดตัวกับวินโดวส์ จึงสุดว่าคุณจะให้มันใหญ่ขนาดไหนถ้าต้องการให้ใหญ่มากก็ให้นับมากครั้งขึ้น

หากต้องการให้ใหญ่คับฟ้าให้เปลี่ยน UNTIL เป็น until eternity คราวนี้จะต้องกดคีย์ Ctrl+Alt+Del เพื่อให้วินโดวส์พยายามยุติการรันโปรแกรมนี้ ซึ่งทำได้บ้างไม่ได้บ้างแต่ส่วนมากไม่ได้ จะต้องกดคีย์ Ctrl+Alt+Del ซ้ำอีกครั้งเพื่อบูตเครื่องใหม่

นั่นคือวินโดวส์จะรันในแบบมัลติทาสกิงหรือไม่ ขึ้นอยู่กับผู้เขียนโปรแกรมด้วย ถ้าเขียนโปรแกรมแบบ IamBig ก็ไม่เป็นมัลติทาสกิง และผมจะขอนำคำว่า IamBig ไปใช้อ้างถึงต่อไปด้วย

การเขียนโปรแกรมแบบ Iam-Big ความจริงแล้วเขียนได้ยากพอสมควร เพราะจะต้องไม่ใช้คีย์บอร์ด, เม้าส์, เมนู, ไดอะล็อกบ็อกซ์ และสิ่งอื่น ๆ ของวินโดวส์ที่จะต้องรับเมสเสจ แต่ก็มีเหมือนกันครับที่ต้องคิดว่าหากเขียนโปรแกรมตามแนวความคิดของเราแล้วจะ IamBig หรือไม่ถึงแม้จะเป็น IamBig เล็ก ๆ ครับ

ท้ายตอน

มาถึงตอนนี้ท่านผู้อ่านคงได้เห็นแล้วว่า ถึงแม้วินโดวส์จะไม่สนับสนุนการสร้างภาพเคลื่อนไหวโดยตรง แต่เราก็พอจะมีวิธีการในการสร้างภาพขึ้นมาใช้งานประกอบบนโปรแกรมที่เขียนได้บ้าง